Ghent University
Faculty of Sciences

Python Programming

listen very carefully

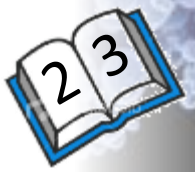
I shall say this only once

Prof. Dr. Peter Dawyndt

peter.dawyndt@ugent.be

@dawyndt

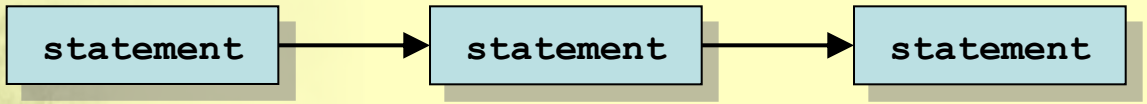
UNIVERSITEIT GENT

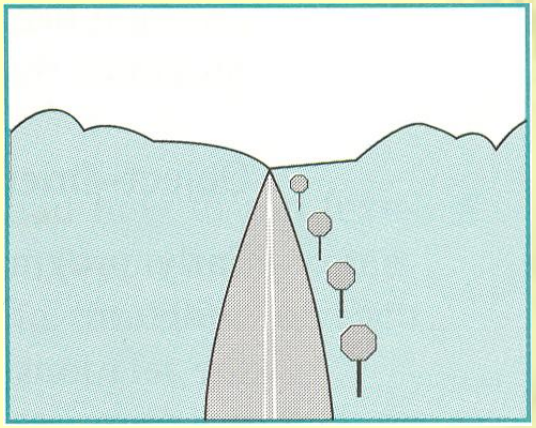


Control structures

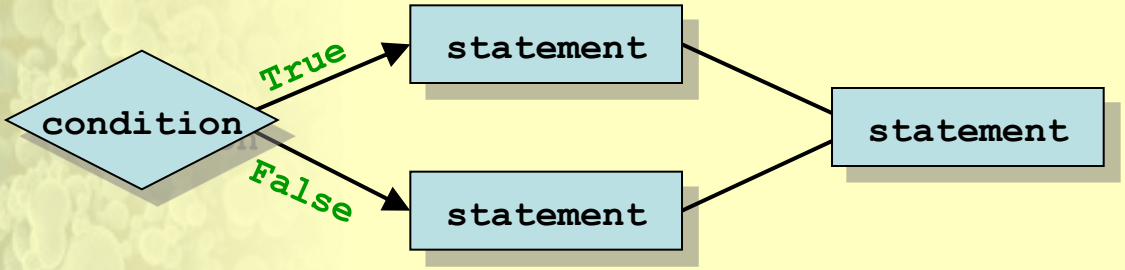
Control structures

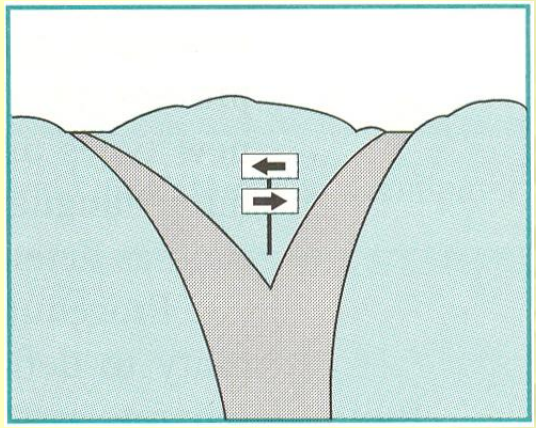
sequence





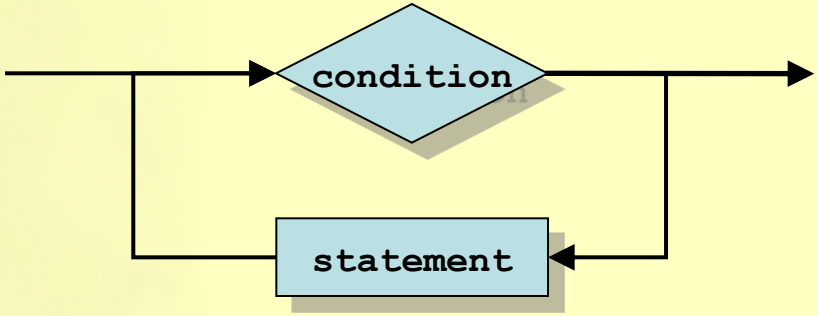
conditional statement (branch)

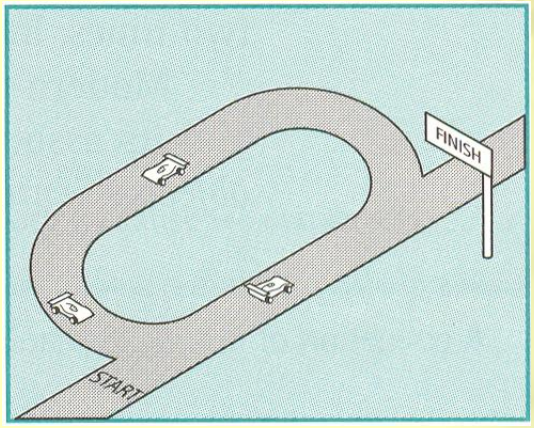




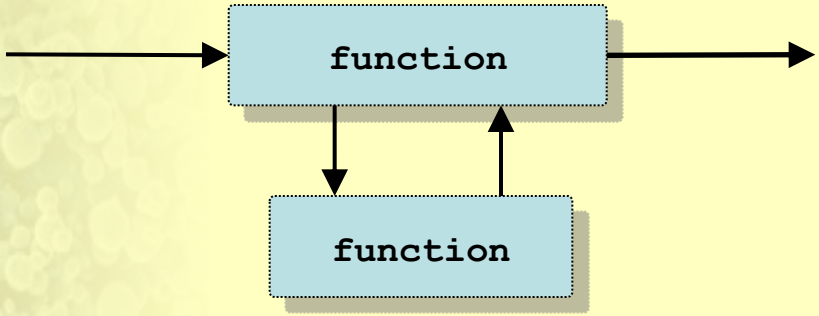
Control structures

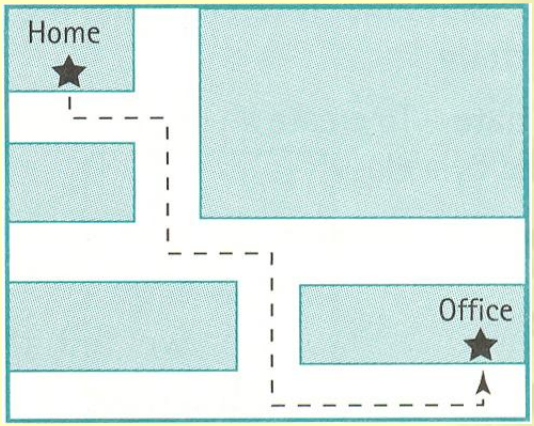
repetitive statement (loop, repetition, iteration)





function (method)



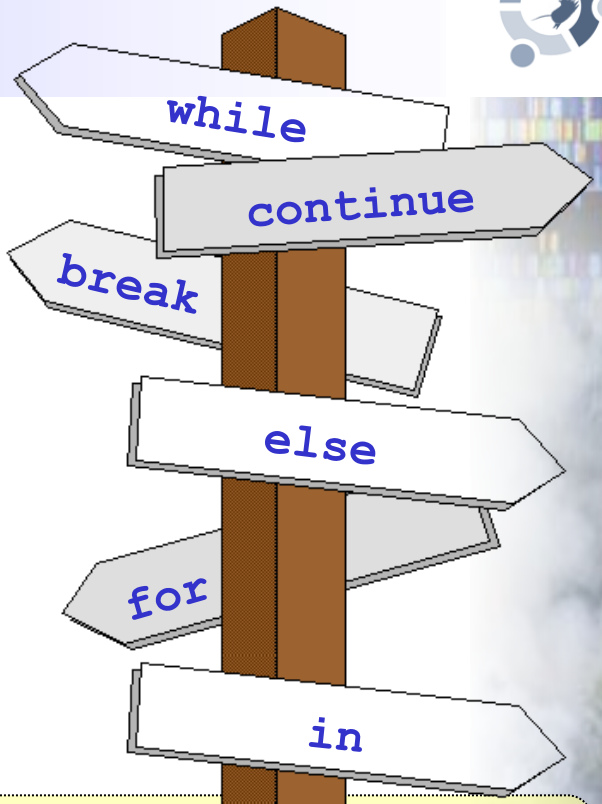


Control structures

- conditional statements
- repetitive statements
- functions

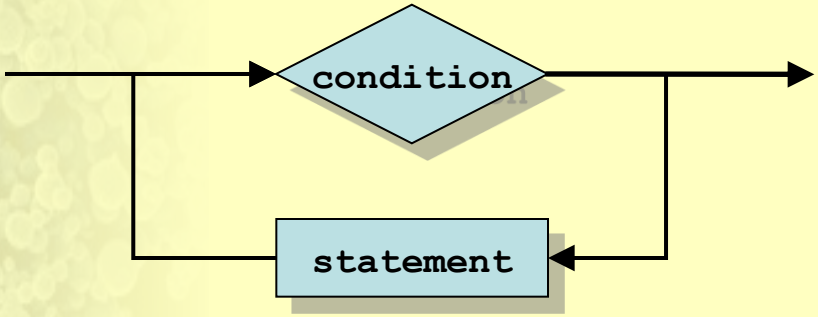
UNIVERSITEIT GENT

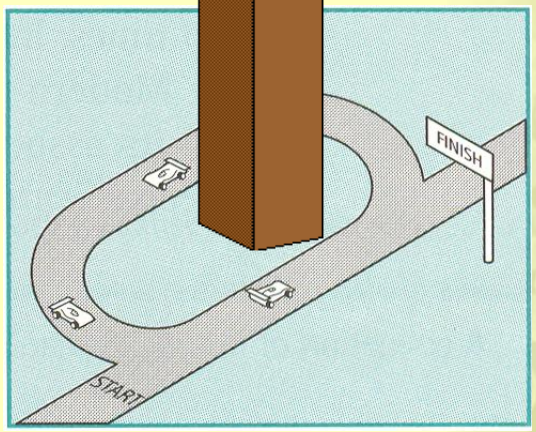
Control structures



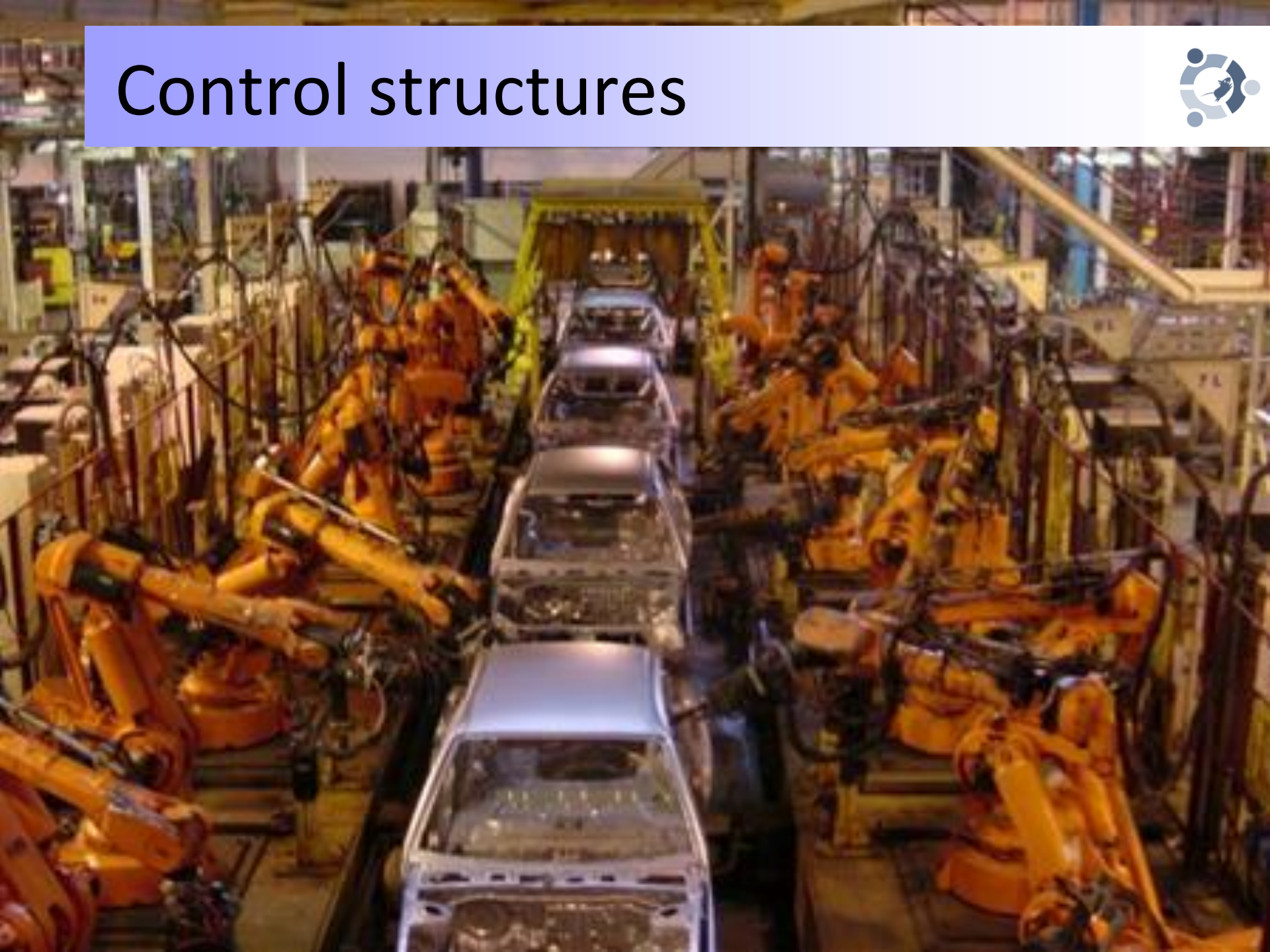
- conditional statements
- repetitive statements
- functions

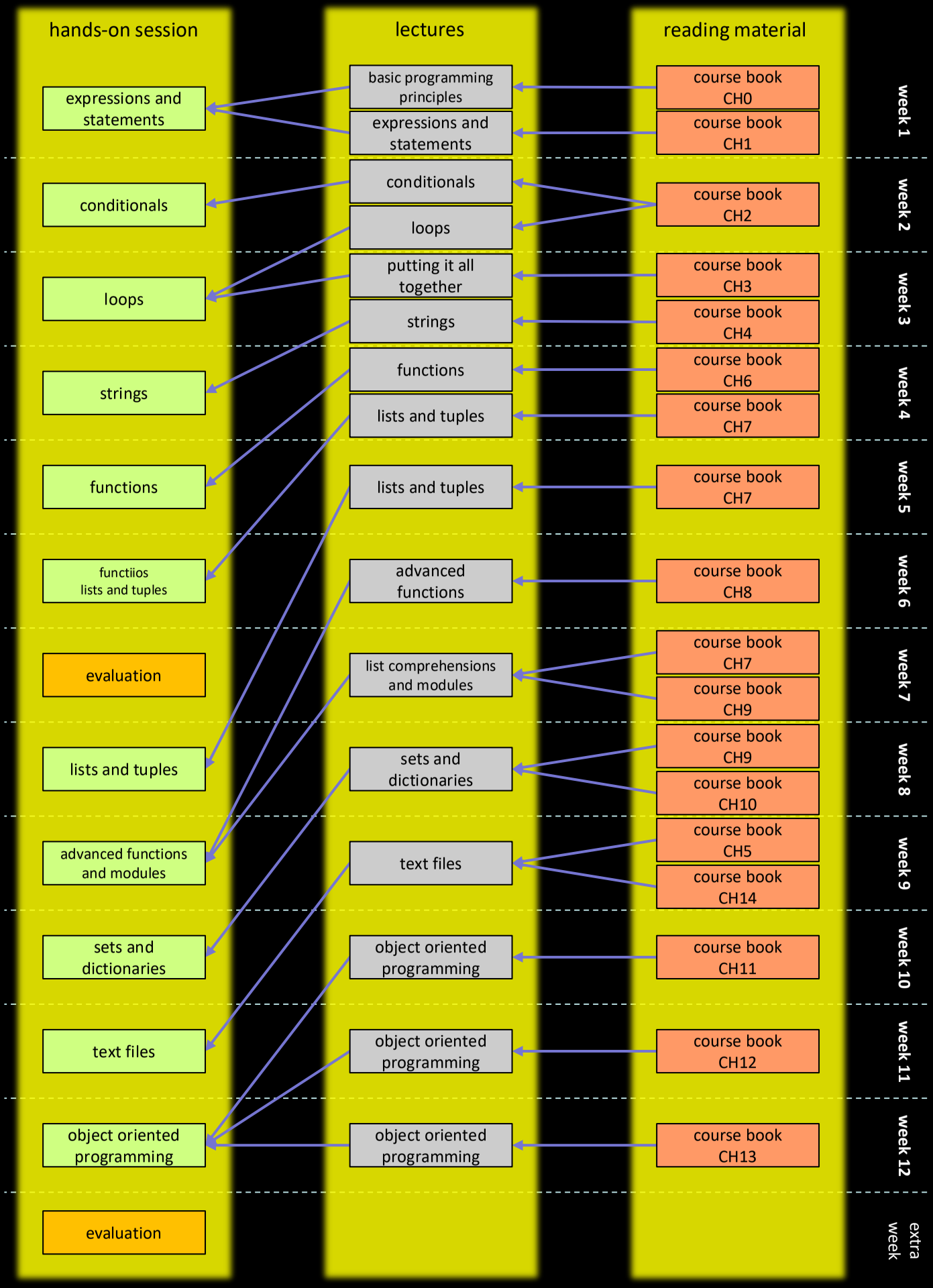
repetitive statement (loop, repetition, iteration)



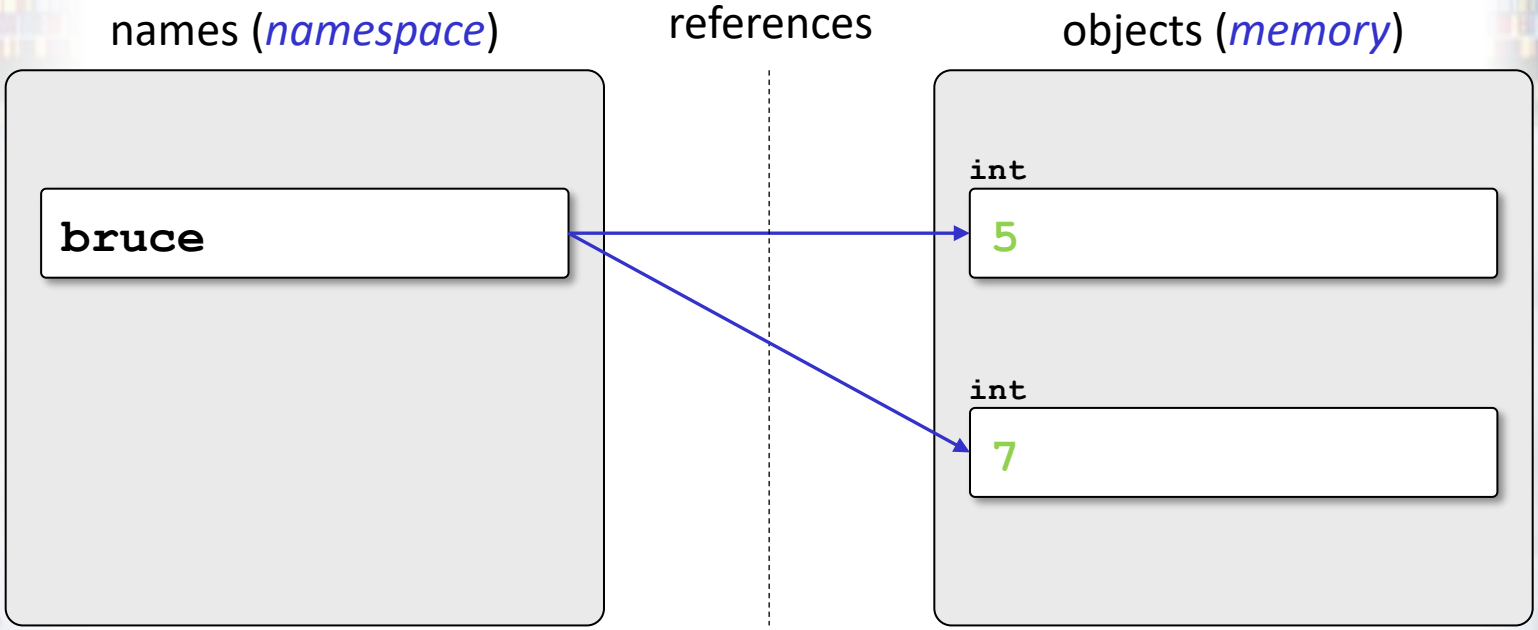


Control structures




hands-on session
lectures
reading material
week 1
week 2
week 3
week 4
week 5
week 6
week 7
week 8
week 9
week 10
week 11
week 12
extra week
expressions and statements
conditionals
loops
strings
functions
functiios lists and tuples
evaluation
lists and tuples
advanced functions and modules
sets and dictionaries
text files
object oriented programming
evaluation
basic programming principles
expressions and statements
conditionals
loops
putting it all together
strings
functions
lists and tuples
lists and tuples
advanced functions
list comprehensions and modules
sets and dictionaries
text files
object oriented programming
object oriented programming
object oriented programming
course book CH0
course book CH1
course book CH2
course book CH3
course book CH4
course book CH6
course book CH7
course book CH7
course book CH8
course book CH7
course book CH9
course book CH9
course book CH10
course book CH5
course book CH14
course book CH11
course book CH12
course book CH13

Multiple assignment



assignment.py

```
bruce = 5
print(bruce, end=' ')
bruce = 7
print(bruce)
```

```
$ python3 assignment.py
5 7
$
```

UNIVERSITEIT GENT

Updating variables

- assignment operator (=)
  - expression on right-hand side is evaluated first
  - then result is assigned to name on left-hand side
  - new value of the variable may depend on the old

```
>>> x = x + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> x = 0
>>> x = x + 1
>>> print(x)
1
```

UNIVERSITEIT GENT

Repetitive statements

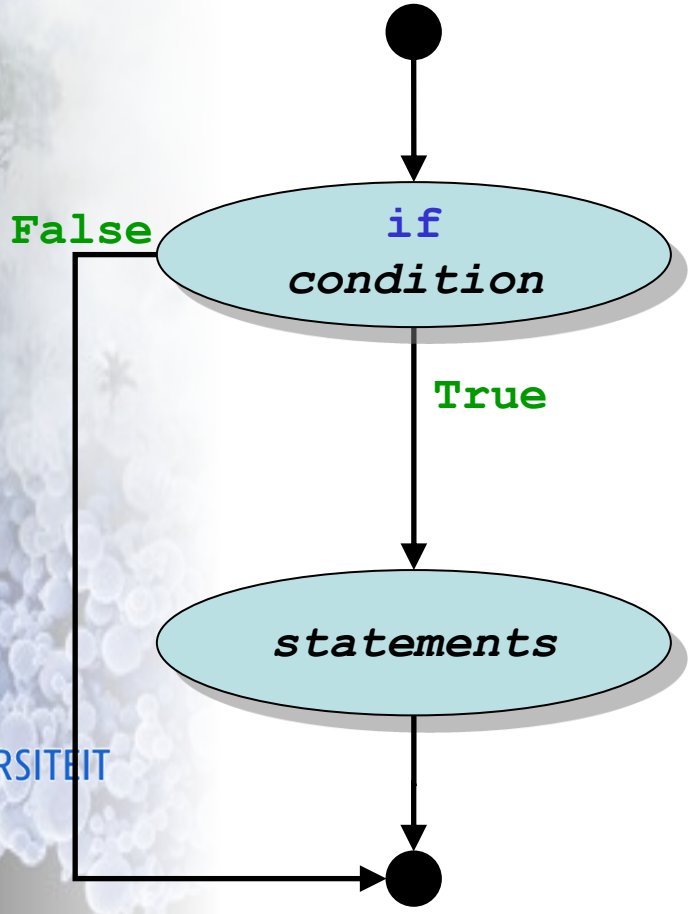
- *repetitive statement* (*loop*)
  - control structure that executes a sequence of statements in succession, and jumps to the first statement after the last statement has been executed
  - each cycle in which instructions of the control structure are executed sequentially is called an *iteration*
- usage
  - manipulation of large amounts of data
    - e.g. processing experimental data in tabular format
  - iterative computation of result
    - e.g. compute cumulative sum
    - e.g. compute average of a series of numbers

UNIVERSITEIT GENT

While loop

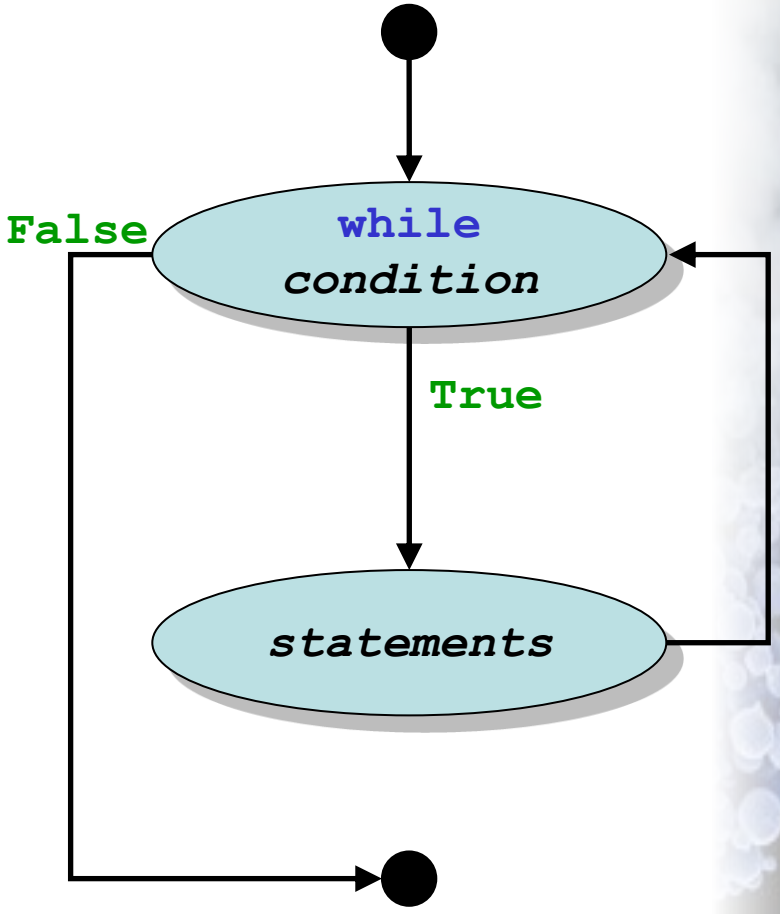
syntax

```
if boolean expression:
    statements
```



syntax

```
while boolean expression:
    statements
```

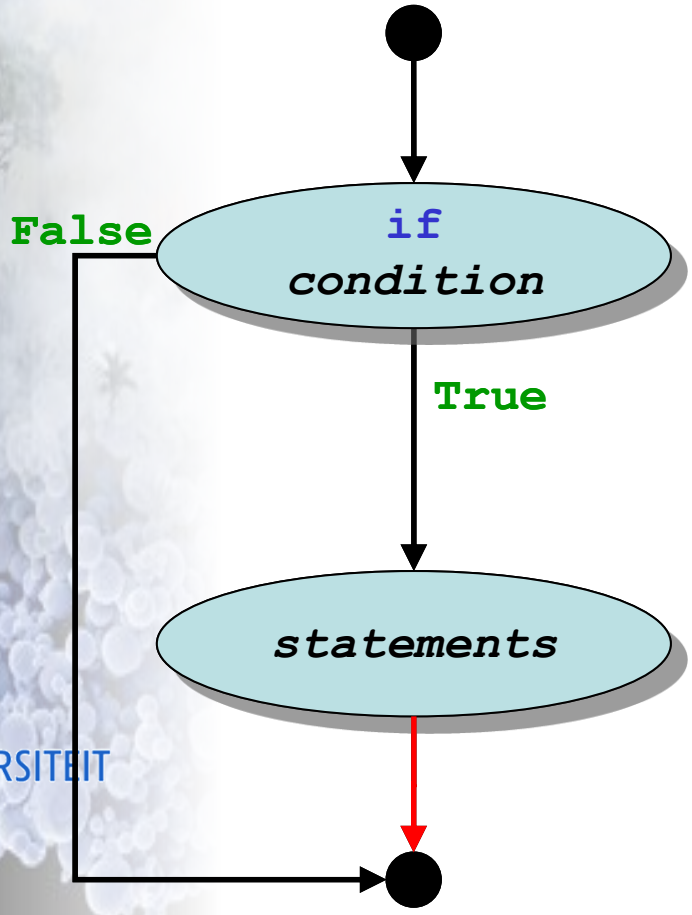


UNIVERSITEIT GENT

While loop

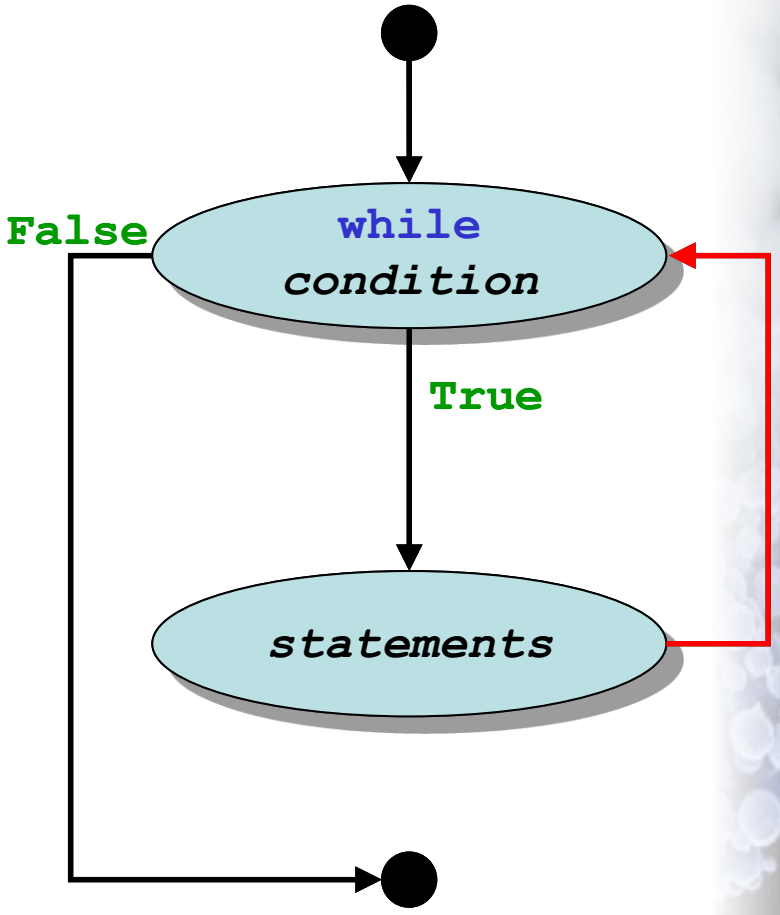
syntax

```
if boolean expression:
    statements
```



syntax

```
while boolean expression:
    statements
```



UNIVERSITEIT GENT

While loop

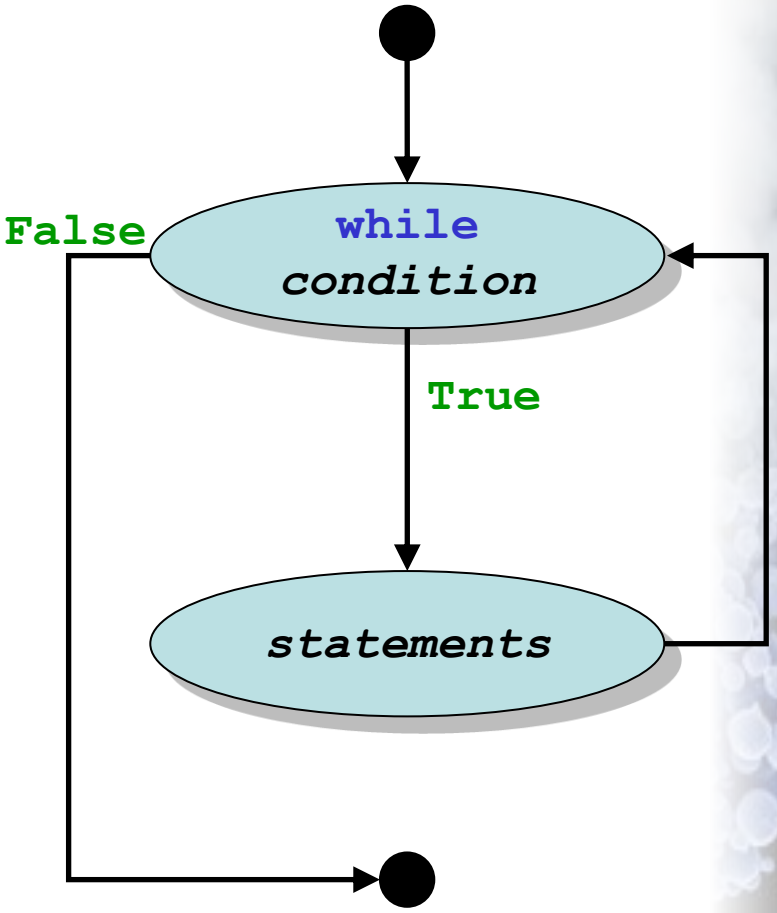
example

countdown.py

```
n = 10
while n > 0:
    print(n)
    n = n - 1
print('Ignition!')
```

syntax

```
while boolean expression:
    statements
```



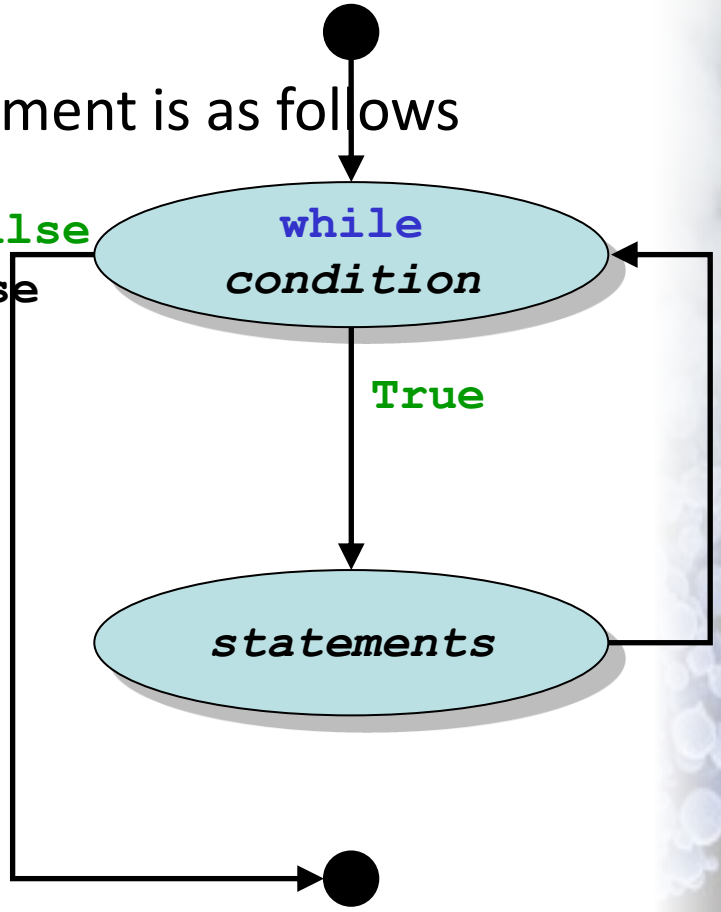
UNIVERSITEIT GENT

While loop

syntax

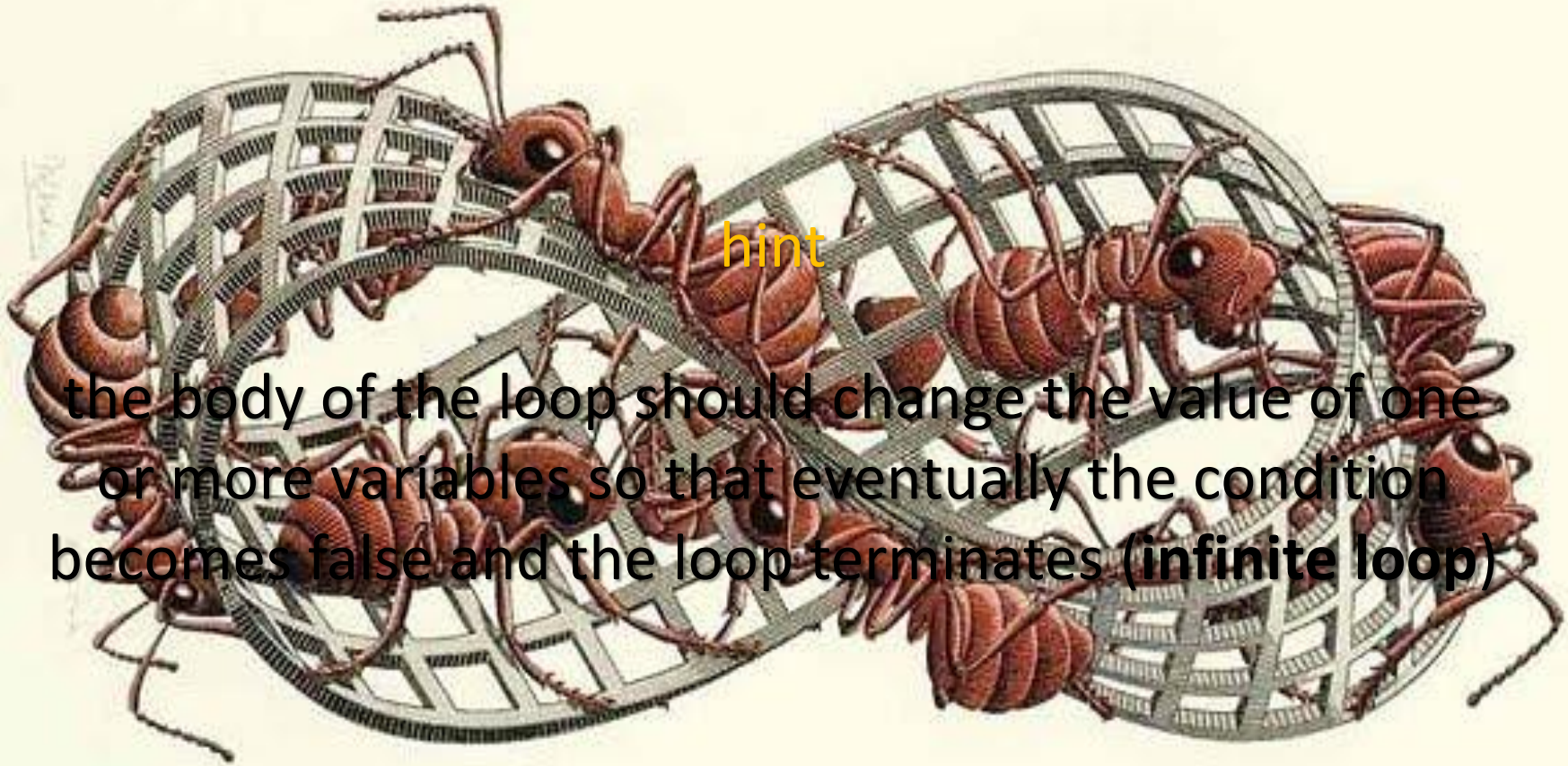
```
while boolean expression:
    statements
```

- flow of execution for a **`while`** statement is as follows
  1. evaluate condition
     - Booleanse expression: **`True`** or **`False`**
  2. condition is **`False`**
     - exit **`while`** statement
     - continue execution at the next statement
  3. condition is **`True`**
     - execute statements in body
     - go back to step 1



UNIVERSITEIT GENT

Infinite loop



hint

the body of the loop should change the value of one or more variables so that eventually the condition becomes false and the loop terminates (**infinite loop**)

Counting digits

digits.py

```
n = int(input('Enter a number: '))

m = n                       # remember value of n
count = 0                   # initialize counter
while m:
    count = count + 1
    m = m // 10

print(n, 'has', count, 'digits.')
```

UNIVERSITEIT GENT

Counting odd digits

odd_digits.py

```
n = int(input('Enter a number: '))

m = n                       # remember value of n
count = 0                   # initialize counter
while m:
    digit = m % 10          # lookup last digit
    if digit % 2:
        count = count + 1
    m = m // 10             # remove last digit

print(n, 'has', count, 'odd digits.')
```

UNIVERSITEIT GENT

Syntactic sugar

- abbreviated syntax for incrementing a variable: **+=**
  - increment value does not have to be 1

```
>>> count = 0
>>> count += 1
>>> count
1
>>> count += 1
>>> count
2
>>> count += 5
>>> count
7
```

UNIVERSITEIT GENT

Syntactic sugar

- abbreviated syntax for incrementing a variable: **+=**
  - increment value does not have to be 1
  - other abbreviations: **-=**, ***=**, **/=**, **//=** and **%=**

```
>>> count = 2
>>> count *= 5
>>> count
10
>>> count -= 4
>>> count
6
>>> count //= 2
>>> count
3
>>> count %= 2
>>> count
1
```

UNIVERSITEIT GENT

Tables

x	0	1 2 3	4 5 6	7 8 9		1 2 3	4 5 6	7 8 9
10	·0000							
11	·0414							
			0607 0645	0682 0719 0755	37	4 7 11	15 19 22	26 30 33
12	·0792	0828 0864 0899	0934 0969		35	4 7 11	14 18 21	25 28 32
			0969 1004	1038 1072 1106	34	3 7 10	14 17 20	24 27 31
13	·1139	1173 1206 1239	1271 1303		33	3 7 10	13 16 20	23 26 30
			1303 1335	1367 1399 1430	32	3 6 10	13 16 19	22 26 29
14	·1461	1492 1523 1553	1584 1614 1644	1673 1703 1732	30	3 6 9	12 15 18	21 24 27
15	·1761	1790 1818 1847	1875 1903 1931	1959 1987 2014	28	3 6 8	11 14 17	20 22 25
16	·2041	2068 2095 2122	2148 2175 2201	2227 2253 2279	26	3 5 8	10 13 16	18 21 23
17	·2304	2330 2355 2380	2405 2430 2455	2480 2504 2529	25	2 5 7	10 12 15	17 20 22
18	·2553	2577 2601 2625	2648 2672 2695	2718 2742 2765	24	2 5 7	10 12 14	17 19 22
19	·2788	2810 2833 2856	2878 2900 2923	2945 2967 2989	22	2 4 7	9 11 13	15 18 20
20	·3010	3032 3054 3075	3096 3118 3139	3160 3181 3201	21	2 4 6	8 11 13	15 17 19
21	·3222	3243 3263 3284	3304 3324 3345	3365 3385 3404	20	2 4 6	8 10 12	14 16 18
22	·3424	3444 3464 3483	3502 3522 3541	3560 3579 3598	19	2 4 6	8 10 11	13 15 17
23	·3617	3636 3655 3674	3692 3711 3729	3747 3766 3784	18	2 4 5	7 9 11	13 14 16
24	·3802	3820 3838 3856	3874 3892 3909	3927 3945 3962	18	2 4 5	7 9 11	13 14 16
25	·3979	3997 4014 4031	4048 4065 4082	4099 4116 4133	17	2 3 5	7 9 10	12 14 15
26	·4150	4166 4183 4200	4216 4232 4249	4265 4281 4298	16	2 3 5	6 8 10	11 13 14
27	·4314	4330 4346 4362	4378 4393 4409	4425 4440 4456	16	2 3 5	6 8 10	11 13 14
28	·4472	4487 4502 4518	4533 4548 4564	4579 4594 4609	15	2 3 5	6 8 9	11 12 14
29	·4624	4639 4654 4669	4683 4698 4713	4728 4742 4757	15	1 3 4	6 7 9	10 12 13
30	·4771	4786 4800 4814	4829 4843 4857	4871 4886 4900	14	1 3 4	6 7 8	10 11 13
31	·4914	4928 4942 4955	4969 4983 4997	5011 5024 5038	14	1 3 4	6 7 8	10 11 13
32	·5051	5065 5079 5092	5105 5119 5132	5145 5159 5172	13	1 3 4	5 7 8	9 10 12
33	·5185	5198 5211 5224	5237 5250 5263	5276 5289 5302	13	1 3 4	5 6 8	9 10 12
34	·5315	5328 5340 5353	5366 5378 5391	5403 5416 5428	13	1 3 4	5 6 8	9 10 12
35	·5441	5453 5465 5478	5490 5502 5514	5527 5539 5551	12	1 2 4	5 6 7	8 10 11
36	·5563	5575 5587 5599	5611 5623 5635	5647 5658 5670	12	1 2 4	5 6 7	8 10 11
37	·5682	5694 5705 5717	5729 5740 5752	5763 5775 5786	12	1 2 4	5 6 7	8 10 11
38	·5798	5809 5821 5832	5843 5855 5866	5877 5888 5899	11	1 2 3	4 6 7	8 9 10
39	·5911	5922 5933 5944	5955 5966 5977	5988 5999 6010	11	1 2 3	4 6 7	8 9 10
40	·6021	6031 6042 6053	6064 6075 6085	6096 6107 6117	11	1 2 3	4 5 7	8 9 10
41	·6128	6138 6149 6160	6170 6180 6191	6201 6212 6222	10	1 2 3	4 5 6	7 8 9
42	·6232	6243 6253 6263	6274 6284 6294	6304 6314 6325	10	1 2 3	4 5 6	7 8 9
43	·6335	6345 6355 6365	6375 6385 6395	6405 6415 6425	10	1 2 3	4 5 6	7 8 9
44	·6435	6444 6454 6464	6474 6484 6493	6503 6513 6522	10	1 2 3	4 5 6	7 8 9
45	·6532	6542 6551 6561	6571 6580 6590	6599 6609 6618	10	1 2 3	4 5 6	7 8 9
46	·6628	6637 6646 6656	6665 6675 6684	6693 6702 6712	9	1 2 3	4 5 5	6 7 8
47	·6721	6730 6739 6749	6758 6767 6776	6785 6794 6803	9	1 2 3	4 5 5	6 7 8
48	·6812	6821 6830 6839	6848 6857 6866	6875 6884 6893	9	1 2 3	4 4 5	6 7 8
49	·6902	6911 6920 6928	6937 6946 6955	6964 6972 6981	9	1 2 3	4 4 5	6 7 8

No. log
π = 3·14159 0·49715 ln $x = \log_e x = (1/M) \log_{10} x$ $(1/M)$ = 2·30259 0·36222
e = 2·71828 0·43429 $\log x = \log_{10} x = M \log_e x$ M = 0·43429 $\bar{1}$·63778

p	1	2	3	4	5	6	7	8	9	10
$\log e^p$	0·4343	0·8686	1·3029	1·7372	2·1715	2·6058	3·0401	3·4744	3·9087	4·3429
$\log e^{-p}$	$\bar{1}$·5657	$\bar{1}$·1314	$\bar{2}$·6971	$\bar{2}$·2628	$\bar{3}$·8285	$\bar{3}$·3942	$\bar{4}$·9599	$\bar{4}$·5256	$\bar{4}$·0913	$\bar{5}$·6571

x	0	1 2 3	4 5 6	7 8 9		1 2 3	4 5 6	7 8 9
53	·7243	7251 7259 7267	7275 7284 7292	7300 7308 7316	8	1 2 2	3 4 5	6 6 7
54	·7324	7332 7340 7348	7356 7364 7372	7380 7388 7396	8	1 2 2	3 4 5	6 6 7
55	·7404	7412 7419 7427	7435 7443 7451	7459 7466 7474	8	1 2 2	3 4 5	6 6 7
56	·7482	7490 7497 7505	7513 7520 7528	7536 7543 7551	8	1 2 2	3 4 5	6 6 7
57	·7559	7566 7574 7582	7589 7597 7604	7612 7619 7627	8	1 2 2	3 4 5	6 6 7
58	·7634	7642 7649 7657	7664 7672 7679	7686 7694 7701	8	1 2 2	3 4 5	6 6 7
59	·7709	7716 7723 7731	7738 7745 7752	7760 7767 7774	7	1 1 2	3 4 4	5 6 6
60	·7782	7789 7796 7803	7810 7818 7825	7832 7839 7846	7	1 1 2	3 4 4	5 6 6
61	·7853	7860 7868 7875	7882 7889 7896	7903 7910 7917	7	1 1 2	3 4 4	5 6 6
62	·7924	7931 7938 7945	7952 7959 7966	7973 7980 7987	7	1 1 2	3 3 4	5 6 6
63	·7993	8000 8007 8014	8021 8028 8035	8041 8048 8055	7	1 1 2	3 3 4	5 6 6
64	·8062	8069 8075 8082	8089 8096 8102	8109 8116 8122	7	1 1 2	3 3 4	5 6 6
65	·8129	8136 8142 8149	8156 8162 8169	8176 8182 8189	7	1 1 2	3 3 4	5 6 6
66	·8195	8202 8209 8215	8222 8228 8235	8241 8248 8254	7	1 1 2	3 3 4	5 6 6
67	·8261	8267 8274 8280	8287 8293 8299	8306 8312 8319	6	1 1 2	2 3 4	4 5 5
68	·8325	8331 8338 8344	8351 8357 8363	8370 8376 8382	6	1 1 2	2 3 4	4 5 5
69	·8388	8395 8401 8407	8414 8420 8426	8432 8439 8445	6	1 1 2	2 3 4	4 5 5
70	·8451	8457 8463 8470	8476 8482 8488	8494 8500 8506	6	1 1 2	2 3 4	4 5 5
71	·8513	8519 8525 8531	8537 8543 8549	8555 8561 8567	6	1 1 2	2 3 4	4 5 5
72	·8573	8579 8585 8591	8597 8603 8609	8615 8621 8627	6	1 1 2	2 3 4	4 5 5
73	·8633	8639 8645 8651	8657 8663 8669	8675 8681 8686	6	1 1 2	2 3 4	4 5 5
74	·8692	8698 8704 8710	8716 8722 8727	8733 8739 8745	6	1 1 2	2 3 4	4 5 5
75	·8751	8756 8762 8768	8774 8779 8785	8791 8797 8802	6	1 1 2	2 3 4	4 5 5
76	·8808	8814 8820 8825	8831 8837 8842	8848 8854 8859	6	1 1 2	2 3 4	4 5 5
77	·8865	8871 8876 8882	8887 8893 8899	8904 8910 8915	6	1 1 2	2 3 4	4 5 5
78	·8921	8927 8932 8938	8943 8949 8954	8960 8965 8971	6	1 1 2	2 3 4	4 5 5
79	·8976	8982 8987 8993	8998 9004 9009	9015 9020 9025	6	1 1 2	2 3 4	4 5 5
80	·9031	9036 9042 9047	9053 9058 9063	9069 9074 9079	5	1 1 2	2 3 3	4 4 5
81	·9085	9090 9096 9101	9106 9112 9117	9122 9128 9133	5	1 1 2	2 3 3	4 4 5
82	·9138	9143 9149 9154	9159 9165 9170	9175 9180 9186	5	1 1 2	2 3 3	4 4 5
83	·9191	9196 9201 9206	9212 9217 9222	9227 9232 9238	5	1 1 2	2 3 3	4 4 5
84	·9243	9248 9253 9258	9263 9269 9274	9279 9284 9289	5	1 1 2	2 3 3	4 4 5
85	·9294	9299 9304 9309	9315 9320 9325	9330 9335 9340	5	1 1 2	2 3 3	4 4 5
86	·9345	9350 9355 9360	9365 9370 9375	9380 9385 9390	5	1 1 2	2 3 3	4 4 5
87	·9395	9400 9405 9410	9415 9420 9425	9430 9435 9440	5	0 1 1	2 2 3	3 4 4
88	·9445	9450 9455 9460	9465 9469 9474	9479 9484 9489	5	0 1 1	2 2 3	3 4 4
89	·9494	9499 9504 9509	9513 9518 9523	9528 9533 9538	5	0 1 1	2 2 3	3 4 4
90	·9542	9547 9552 9557	9562 9566 9571	9576 9581 9586	5	0 1 1	2 2 3	3 4 4
91	·9590	9595 9600 9605	9609 9614 9619	9624 9628 9633	5	0 1 1	2 2 3	3 4 4
92	·9638	9643 9647 9652	9657 9661 9666	9671 9675 9680	5	0 1 1	2 2 3	3 4 4
93	·9685	9689 9694 9699	9703 9708 9713	9717 9722 9727	5	0 1 1	2 2 3	3 4 4
94	·9731	9736 9741 9745	9750 9754 9759	9763 9768 9773	5	0 1 1	2 2 3	3 4 4
95	·9777	9782 9786 9791	9795 9800 9805	9809 9814 9818	5	0 1 1	2 2 3	3 4 4
96	·9823	9827 9832 9836	9841 9845 9850	9854 9859 9863	4	0 1 1	2 2 2	3 3 4
97	·9868	9872 9877 9881	9886 9890 9894	9899 9903 9908	4	0 1 1	2 2 2	3 3 4
98	·9912	9917 9921 9926	9930 9934 9939	9943 9948 9952	4	0 1 1	2 2 2	3 3 4
99	·9956	9961 9965 9969	9974 9978 9983	9987 9991 9996	4	0 1 1	2 2 2	3 3 4

Tables

n	n^2	n^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

n^2	n^3
1	1
4	8
9	27
16	64
25	125
36	216
49	343
64	512
81	729
100	1000

UNIVERSITEIT GENT

Tables

```
# print table of powers
print(1, end=' ')
print(1 ** 2, end=' ')
print(1 ** 3)
```

n	n^2	n^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

UNIVERSITEIT GENT

Tables

```
# print table of powers
print(1, 1 ** 2, 1 ** 3)
```

n	n^2	n^3
1	**1**	**1**
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

UNIVERSITEIT GENT

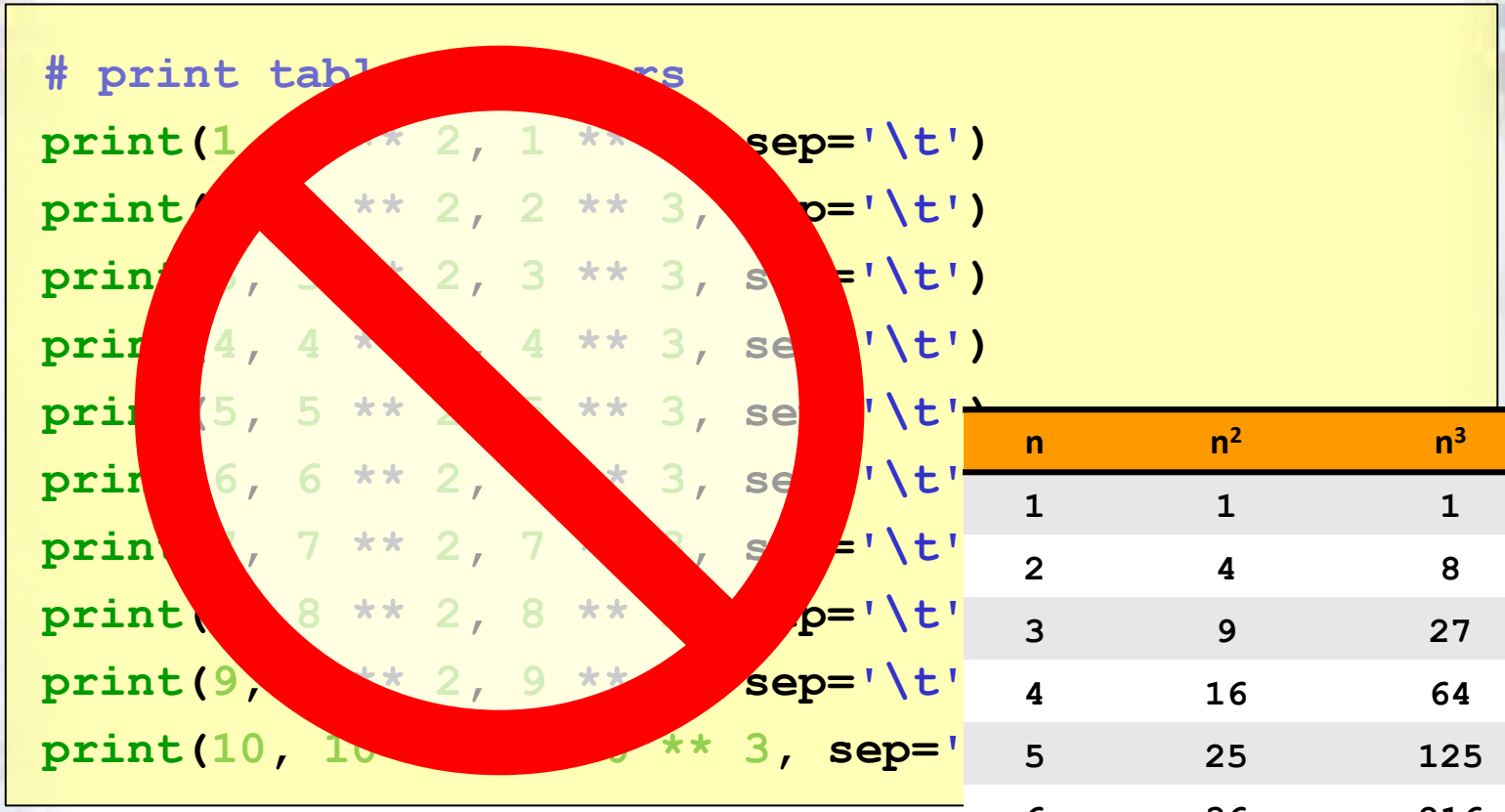
Tables

```
# print table of powers
print(1, '\t', 1 ** 2, '\t', 1 ** 3)
```

n	n^2	n^3
1	**1**	**1**
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

UNIVERSITEIT GENT

Tables



n	n^2	n^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

code duplication (DRY)

UNIVERSITEIT GENT

Tables

```
# print table of powers
n = 1
while n <= 10:
    print(n, n ** 2, n ** 3, sep='\t')
    n += 1
```

n	n^2	n^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

UNIVERSITEIT GENT

Tables

```
# print table of powers
n = 1
while n <= 1000:
    print(n, n ** 2, n ** 3, sep='\t')
    n += 1
```

n	n^2	n^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

UNIVERSITEIT GENT

Tables

```
# print table of powers
n = 1
while n <= 1000:
    print(n, end='\t')
    print(n ** 2, end='\t')
    print(n ** 3)
    n += 1
```

n	n^2	n^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

UNIVERSITEIT GENT

Tables

```
# print table of powers
n = 1
while n <= 1000:
    print(n ** 1, end='\t')
    print(n ** 2, end='\t')
    print(n ** 3)
    n += 1
```

n	n^2	n^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

UNIVERSITEIT GENT

Tables

```
# print table of powers
n = 1
while n <= 1000:
    m = 1
    while m <= 3:
        print(n ** m, end='\t')
        m += 1
    print()
    n += 1
```

n	n^2	n^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

UNIVERSITEIT GENT

Tables

```
# print table of powers
n = 1
while n <= 1000:
    m = 1
    while m <= 10:
        print(n ** m, end='\t')
        m += 1
    print()
    n += 1
```

n	n^2	n^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

UNIVERSITEIT GENT

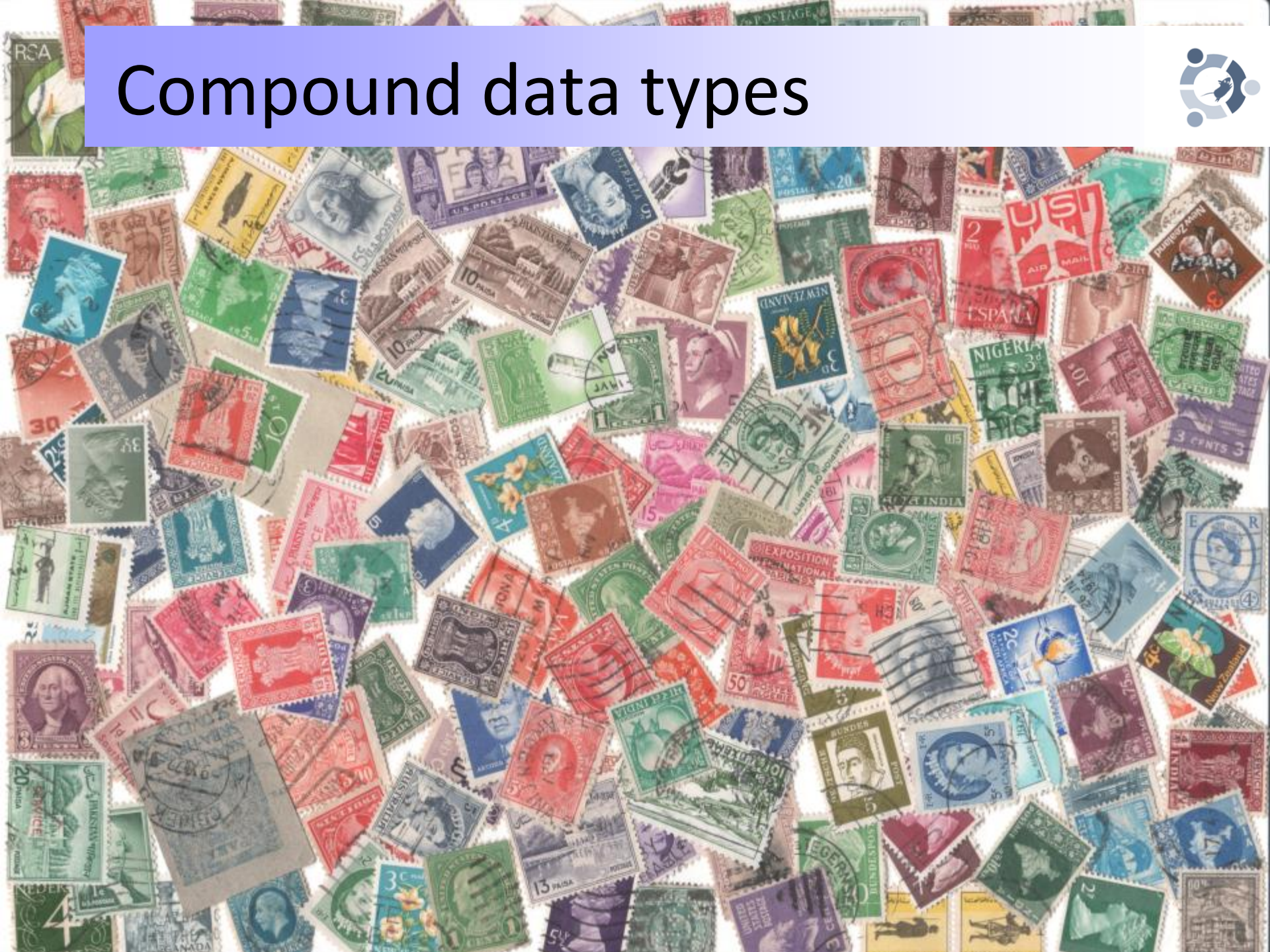
Multiplication table

```
# 10 x 10 multiplication table
n = 1
while n <= 10:
    m = 1
    while m <= 10:
        print(n * m, end='\t')
        m += 1
    print()
    n += 1
```

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

UNIVERSITEIT GENT

Compound data types



Compound data types

- **compound data type**: data type composed of elements that are themselves objects of another data type
  - string
  - list
  - tuple
  - set

```
>>> s = 'spam'
>>> l = [2, 4, 6, 8, 10, 8, 6, 4, 2]
>>> t = ('b', 'a', 'n', 'a', 'n', 'a')
>>> v = {'b', 'a', 'n', 'a', 'n', 'a'}
>>> v
{'a', 'b', 'n'}
```

UNIVERSITEIT GENT

Iterators

- **iterator**: object associated with compound type that can be used to traverse elements of the compound type one by one

```
>>> s = 'spam'
>>> iterator = iter(s)
>>> next(iterator)
's'
>>> next(iterator)
'p'
>>> next(iterator)
'a'
>>> next(iterator)
'm'
>>> next(iterator)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

UNIVERSITEIT GENT

For loop

- **iterator**: object associated with compound type that can be used to traverse elements of the compound type one by one
- **for loop**
  - uses iterator to traverse elements of compound type
  - during each iteration step, next element is assigned to variable

```
>>> word = 'spam'
>>> for letter in word:
...     print(letter)
...
s
p
a
m
>>>
```

UNIVERSITEIT GENT

Range function

- creates an iterator of integers
  - **`range(n)`** : 0, 1, …, n-1
  - **`range(m, n)`** : m, m+1, …, n-1
  - **`range(m, n, p)`** : m, m+p, …, m+*k*p
    - with *k* following the condition m+kp < n ≤ m+(*k*+1)p

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(3, 10))
[3, 4, 5, 6, 7, 8, 9]
>>> list(range(3, 10, 2))
[3, 5, 7, 9]
>>> list(range(10, 0, -1))
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Table of powers

```
# print table of powers
for n in range(1, 11):
    for m in range(1, 4):
        print(n ** m, end='\t')
    print()
```

n	n^2	n^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

UNIVERSITEIT GENT

Table of powers

```
# print table of powers
for n in range(10):
    for m in range(3):
        print((n + 1) ** (m + 1), end='\t')
    print()
```

n	n^2	n^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

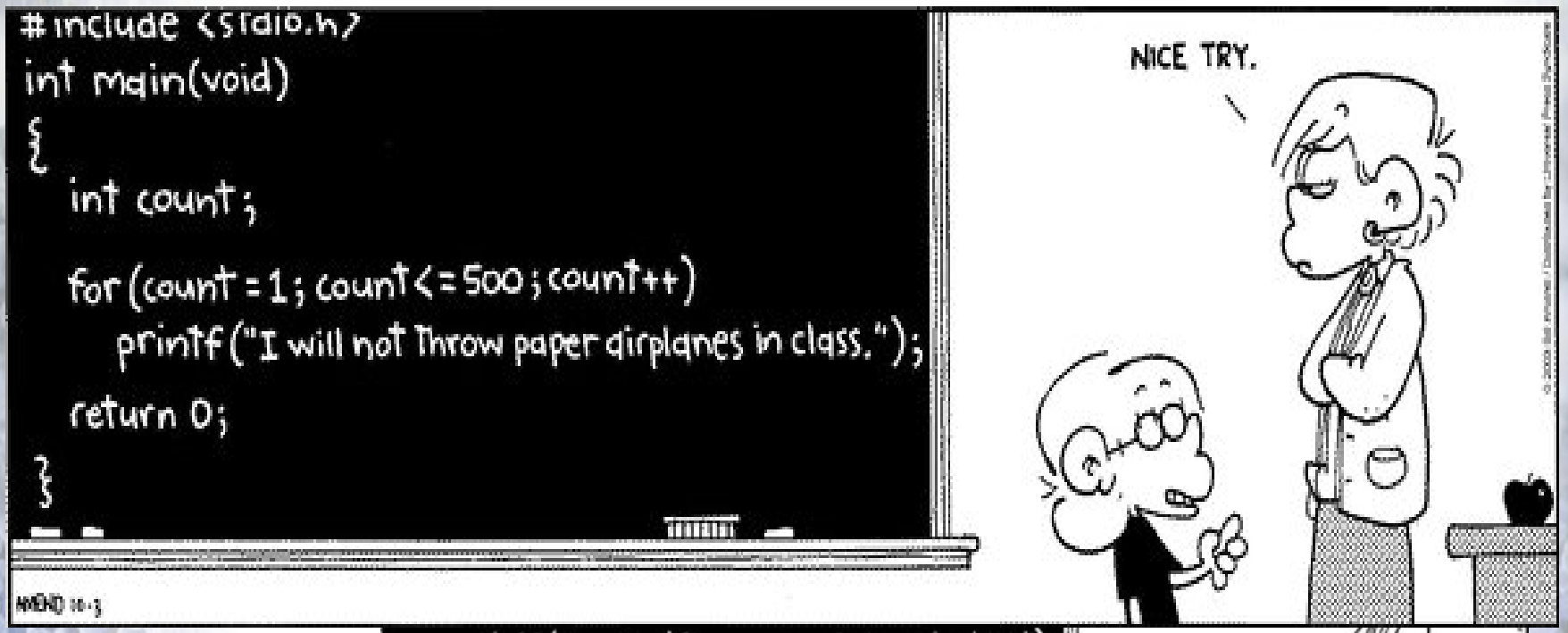
UNIVERSITEIT GENT

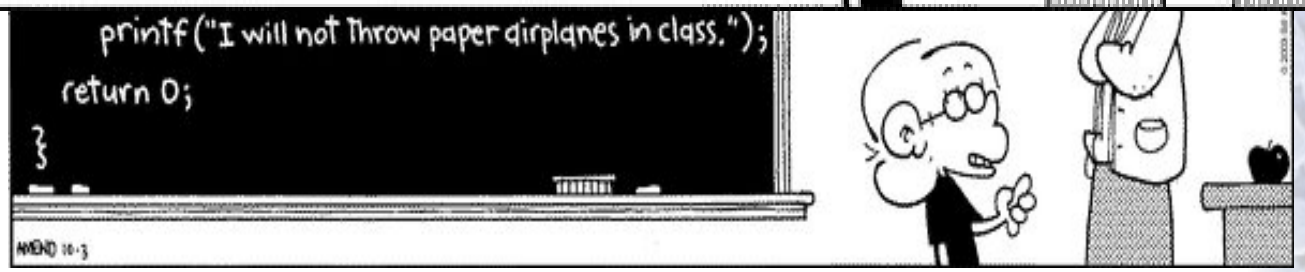
Multiplication table

```
# 10 x 10 multiplication table
for n in range(10):
    for m in range(10):
        print((n + 1) * (m + 1), end='\t')
    print()
```

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

For or while ?

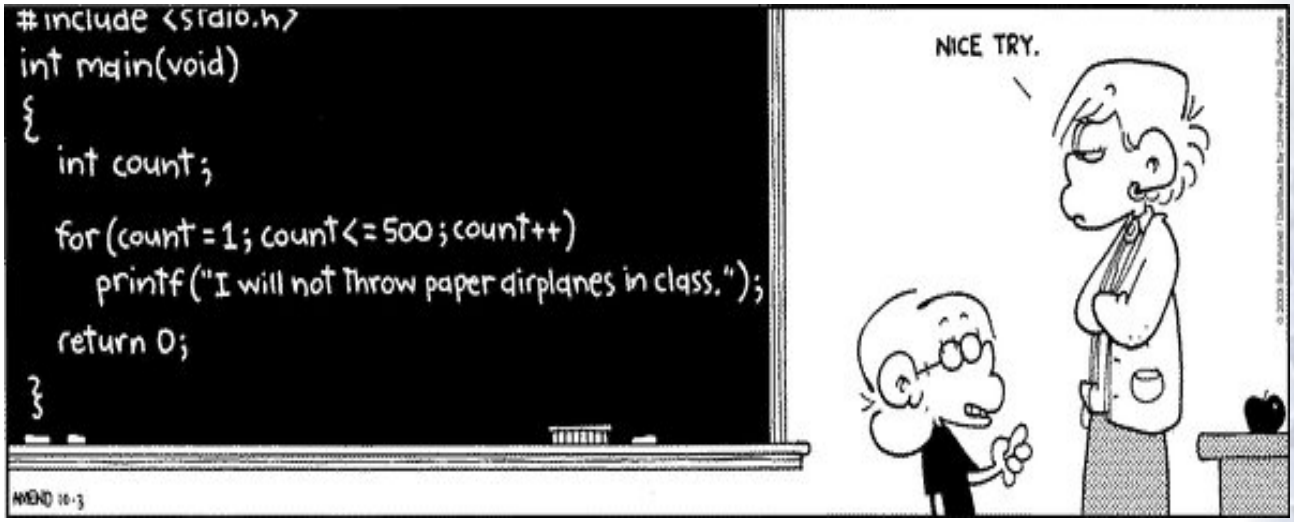




UNIVERSITEIT GENT

For or while ?

- for loops
  - number of iterations is **fixed** at the start of the loop
  - traversing elements of collection type using an iterator
- while loops
  - each iteration has an associated **condition** that determines whether or not a new iteration should be started



UNIVERSITEIT GENT

For or while ?

`for`

number of laps is
fixed for a F1 race

`while`

MTX races take
20 minutes + 2 laps

UNIVERSITEIT GENT

Homework (next lecture)

- course book
  - read chapter 4 (strings)
  - read chapter 6 (functions)
- read problem description of demo exercises
  - Caesar cipher
  - Letter soup
  - Toothpicks
  - Emirp
  - Sanger sequencing
  - Mobile synonyms

Homework (hands-on sessions)

- mandatory exercises of series 03 (control loops) (deadline Tuesday, October 14, 2025, 22:00)
  - ISBN (demo)
  - Chaos
  - German tanks
  - Monkeys and coconuts
  - Pythagorean triples
  - Hitchhikers problem

Questions or remarks ?

UNIVERSITEIT GENT

The sky is the limit …

"Bis repetita placent"

— Horatius, Ars Poetica 365

UNIVERSITEIT GENT